PATENT APPLICATION

HARDWARE IMPLEMENTATION OF AN N-WAY DYNAMIC LINKED LIST

Inventor: John Marino
108 Whits Road
Mountain View, CA 94040

Assignee: Network Equipment Technologies, Inc.
6900 Paseo Padre Parkway
Fremont, CA 94555

Entity: Large

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111-3834
Tel: 415-576-0200

HARDWARE IMPLEMENTATION OF AN N-WAY DYNAMIC LINKED LIST

CROSS REFERENCE TO RELATED APPLICATIONS

[01]   This is a non-provisional application which claims priority from U.S. Provisional

5   Application No. 60/455,624, filed March 17, 2003, and is herein incorporated by reference for all purposes.

BACKGROUND OF THE INVENTION

[02]   The present invention is generally related to logic circuits, and in particular to logic

10   for implementing a linked list mechanism.

[03]   A linked list is a data structure used in application programs to store data that is dynamic in nature. A linked list comprises two basic components: a data element and a pointer to the next element in the linked list. In software, a linked list can grow simply by allocating memory from the operating system (OS) for each new node (entry) in the linked

15   list. Conversely, a linked list can shrink by freeing the memory allocated for one or more nodes (entries) and returning it to the OS. The OS typically provides a memory management component to manage the allocation and deallocation of memory for a application.

[04]   In more specialized systems, memory resources are usually more limited and therefore must be allocated more efficiently. For example, there may not be sufficient

20   memory capacity to provide for a sophisticated OS. Nonetheless, link list capability may still be needed for applications running on these specialized systems. One approach is for the application to allocate a block of memory and to manage the link lists from that memory. When more than one link list is needed, there are two approaches for allocating the link lists: statically or dynamically. Statically allocating memory for N linked lists is not an efficient

25   allocation strategy. Some lists may grow beyond their statically allocated size, while other lists may be underutilized. Since the lists are statically allocated, memory from the underutilized lists cannot be allocated to lists which have grown beyond their predefined sizes. Dynamic allocation from the block of memory requires more processing time; however, memory usage is more efficient.

30

BRIEF SUMMARY OF THE INVENTION

[05] In accordance with the present invention an N-way linked list capability is provided using a dynamic allocation strategy provided in hardware. A first memory stores a pointer to the beginning and end of a linked list. The first memory stores many such pointers, thus providing for many linked lists. A second memory stores the data component of the entries (nodes) which comprise a linked list. An additional memory stores information that indicates the order of the entries of a linked list. Still another memory stores a list of available memory locations in the second memory. Suitable control circuitry is provided to perform dynamic allocation and deallocation of memory for entries in the linked lists. Each entry can be allocated to any linked list, and any entry can be freed from a linked list and subsequently reallocated another linked list. The invention avoids "dead spots" that can arise when linked lists are statically defined, where entries of an underutilized linked list cannot be used to grow a linked list that has reached its limit.

BRIEF DESCRIPTION OF THE DRAWINGS

[06] The features and advantages of the present invention will be more clearly understood from the following description taken in conjunction with the accompanying drawings in which:

Fig. 1 is a high level block diagram illustrating an embodiment of the present invention;

Fig. 2 highlights the processing steps for a write operation in accordance with an embodiment of the present invention;

Fig. 3 illustrates the data flow during a write operation;

Fig. 4 highlights the address and data buses of Fig. 1 during a write operation;

Fig. 5 highlights the processing steps for a read operation in accordance with an embodiment of the present invention;

Fig. 6 illustrates the data flow during a read operation;

Fig. 7 shows the address and data buses shown in Fig. 1 that are involved during a read operation; and

Figs. 8A - 8J show snapshots of the various memory components during a sequence of writes and a read operation in accordance with the invention.

2

DETAILED DESCRIPTION OF THE INVENTION

[07]   Fig. 1 illustrates a typical embodiment of the present invention. A hardware queue 100 according to the embodiment exemplar shown in Fig. 1 includes a register file 102 comprising three register sets. A tail of queue (TOQ) register set 112, a head of queue (HOQ) register set 114, and link length (LL) register set 116. In the discussion which follows, the same reference numeral will be used to refer to both a register set and to an individual register in the register set, depending on the context of the discussion. Suitable control logic 102a provides access to the individual registers comprising the register sets for data read and write operations.

[08]   The particular implementation of the register file 102 shown in Fig. 1 provides for 16 queues. Each queue is a linked list, and has an associated head pointer to the head entry in the linked list, a tail pointer to the associated tail entry in the linked list, and a length indicator indicating the number of entries in the linked list. Thus, each register triplet in the register file comprising an HOQ register, a TOQ register, and a LL register represents a queue. Each register set 112, 114, 116 therefore comprises sixteen registers, to provide for 16 queues. Information stored in the HOQ and TOQ registers represents an address. In the implementation shown, addressing is assumed to be 12-bit addressing to provide 4K (1K = 1024) of address space. Therefore, each register set is 12 bits wide, including the LL register in the case where a queue grows to 4K entries. By convention, the queues are numbered from 0 - 15, using notation such as Q0 - Q15 to identify each queue. The hardware shown in Fig. 1 therefore is said to implement a 16-way dynamic link list. The "dynamic" aspect of the invention will be discussed below.

[09]   A buffer RAM 104 serves as the data space for the queues. Each datum in this particular implementation is assumed to be 128 bits wide, although of course, any data size can be accommodated. Each memory location in the buffer RAM represents the data field of an entry in a linked list. The buffer RAM is therefore a 4K x 128 bit memory, for a maximum number of 4K entries. The figure also shows control logic 104a is provided to access memory in the buffer RAM, for reading an accessed memory location and for writing an accessed memory location.

[10]   A next pointer RAM 106 is provided by a 4K x 12 bit memory. The next pointer RAM stores addresses of memory in the buffer RAM 104, hence the 12 bit data width of the memory. The next pointer RAM will be discussed further, but briefly, addresses stored in this memory are organized in a manner that indicate the link order of the entries of a queue.

Control logic 106a is provided to access memory in the next pointer RAM, for reading an accessed memory location and for writing an accessed memory location.

[11]   A free pointer RAM 108 is provided by a 4K x 12 bit memory. This memory stores addresses of memory in the buffer RAM 104. Since the buffer RAM contains 4K memory locations, this particular implementation of the free pointer RAM contains is at least a 4K x 12 bit memory. As the name implies, the free pointer RAM contains the addresses in the buffer RAM of those memory locations that have not been allocated to an entry in a queue. Though this memory is referred to as a RAM (random access memory), the free pointer memory is accessed in FIFO (first-in, first-out) fashion. Data is "pushed", i.e., written into the free pointer RAM at the bottom of the queue, logically shown in the figure at 108b. Data is read from the free pointer RAM ("popped") at the top of the queue, logically represented as 108a.

[12]   Controller logic comprises decoder logic 134 and control logic 132. The decoder logic includes an input bus 172 that is configured to receive incoming data comprising a queue identifier (QID), an operation (i.e., read, write), and data to written in the case of a write operation. The decoder logic produces a suitable control signal(s) 161 for controlling the register file 102 to select a register triplet based on the QID. Another control signal 165 is provided to the control logic 132 to indicate to the control logic whether an incoming operation is a write operation to one of the queues or a read operation from one of the queues.

[13]   The control logic 132 is configured to generate various control signals for synchronizing and otherwise controlling the actions of the other components shown in the figure. A control signal 162 controls the register file 102 to increment or decrement the content of one of the LL registers 116, selected based on the QID. It can be appreciated that the control logic 102a associated with the register file can be further configured to perform the increment or decrement operation on the selected LL register. A control signal 163 controls whether data is written to the buffer RAM 104 or read from the buffer RAM. A control signal 164 controls the free pointer RAM 108 to perform a "push" operation of a "pop" operation, depending on the incoming operation detected by the decoder logic 134.

[14]   An address bus 153 is coupled to an output of the HOQ register set 114 and to an output of the TOQ register set 112. The address bus feeds into the control logic 104a of the buffer RAM 104 and to the control logic 106a of the next pointer RAM 106. A register can be selected from among the HOQ register set and the TOQ register set, and the data contained in the selected register can be driven onto the address bus to feed the data as

4

address to the buffer RAM and to the next pointer RAM. In the particular implementation shown in Fig. 1, the address bus is a 12-bit bus.

[15] A 128-bit data bus 152 feeds data from the decoder logic 134 to the buffer RAM 104. The data is then written to an accessed memory location in the buffer RAM. Incoming data received by the decoder logic includes a data component when the operation is a write operation. It can be appreciated that the decoder logic can be configured to obtain the data associated with a write operation and to provide that data on the data bus.

[16] A data bus 155 feeds data that is popped from the free pointer RAM 108 to the next pointer RAM 106 and to the TOQ register set 112. Data from the free pointer RAM is stored to an accessed memory location in the next pointer RAM. Similarly, data from the free pointer RAM is stored in the TOQ register that is selected based on the incoming QID.

[17] A data bus 154 feeds data from an accessed location in the next pointer RAM 106 to the free pointer RAM 108 and to the HOQ register set 112. Data from the next pointer RAM is pushed onto the bottom 108b of the queue provided by the free pointer RAM. Data from the next pointer RAM is stored to one of the HOQ registers determined by the QID.

[18] A data bus 151 feeds data from a selected LL register 116 to the control logic. The LL register that is selected is based on the incoming QID.

[19] The following initial conditions are set when the circuit shown in Fig. 1 is reset. The register file 102 can be initialized as follows, where Qn is the nth queue, H[T]n is the content of the H[T]OQ register 114 [112] of the nth queue and Ln is the LL register of the nth queue:

        Q0:   T0 = 0, H0 = 0, LL = 0
        Q1:   T1 = 1, H1 = 1, LL = 0
        Q2:   T2 = 2, H2 = 2, LL = 0
               :
               :
        Q14:   T14 = 14, H14 = 14, LL = 0
        Q15:   T15 = 15, H15 = 15, LL = 0

[20] As will be explained below, the convention for writing to a queue in accordance with this embodiment of the invention is to assume that the TOQ register always contains the address of the memory location in the buffer RAM 104 to which the data will be written. Hence, the first entry for Q0 is location 0 in the buffer RAM. The first entry in Q1 is location 1 in the buffer RAM, and so on.

[21] Recall that the free pointer RAM 108 is accessed as a queue. There is the notion of the "top" of the queue and the "bottom" (or tail) of the queue. The free pointer RAM stores the list of addresses of memory locations in the buffer RAM 104 that have not been allocated

5

to a queue. The top of the queue is a memory location in the free pointer RAM that contains the first memory location in the buffer RAM that will be allocated. When a memory location in the buffer RAM is freed (e.g., as the result of a read operation), the address of the freed memory location is pushed to the bottom of the queue in the free pointer RAM.

[22] The first sixteen memory locations in the 4K buffer RAM 104 have already been allocated to their respective queues. Thus, 4K less 16 entries remain in the buffer RAM for allocation. The free pointer RAM 108 is initialized as follows:

$$\text{top} \quad \rightarrow \quad 16$$
$$17$$
$$18$$
$$\vdots$$
$$\text{bottom} \rightarrow \quad 2^{12} - 1$$

The top of the queue contains the memory location 16 and the bottom of the queue contains the memory location $2^{12} - 1$. When the free pointer RAM "popped", the address 16 will be produced and the top of the queue will then contain the memory location 17, and so on. When a freed memory location is "pushed" onto the queue, then the bottom of queue will contain the pushed memory location.

[23] It can be appreciated that the register file 102 can be initialized with different values. The free pointer RAM 108 would have to be initialized accordingly to reflect the memory locations that are initially assigned to the queues. It can be further appreciated that the order of the memory locations initially stored in the free pointer RAM can be some other order.

[24] The buffer RAM 104 does not require initialization, since all queues are initially empty and the first operation that makes sense for each queue after the initialization is a write operation. Similarly, the next pointer RAM 106 does not require initialization. Since each queue is of length zero at initialization, there is no "next" entry to speak of at that time. Nonetheless, it may be desirable to explicitly set all memory locations in the buffer RAM and the next pointer RAM to some value, e.g., "0" or all "1"s.

[25] Refer now to Figs. 2, 3, and 4 for a discussion of a write operation in accordance with the embodiment of the invention shown in Fig. 1. Fig. 2 is a high level flow chart, highlighting the steps comprising a write operation on a queue. Fig. 3 graphically represents the data flow among the elements of the circuit shown in Fig. 1 during a write operation. Fig. 4 highlights the data flow in the circuit diagram of Fig. 1 during a write operation. It can be appreciated that the decoder logic 134 and the control logic 132 can be configured to perform the operations that will be discussed in connection with a write operation.

6

[26]   When a data write request is communicated to the circuit of Fig. 1, the request comprises a queue identifier (QID), the data to be written, and an indication of the operation to be performed, i.e., a write operation. In a step 202, the register file 102 is accessed with the QID to access the TOQ register 112 corresponding to the QID. This is illustrated in Fig. 3 as the production of the tail register, identified by the step reference 202. The content of the tail register is a memory location in the buffer RAM 104 into which the data will be stored. Thus, in a step 204, a memory location in the buffer RAM is accessed and the data is written into the accessed memory location. Fig. 3 shows the tail register "pointing" to the accessed memory location.

[27]   Then in a step 206, the top of the queue in the free pointer RAM 108 is popped. This is indicated in Fig. 3 by the production of NextValid which represents a location in the buffer RAM that will be used to store the next data value that is written to the specified queue.

[28]   In a step 208, the content of the selected TOQ register 112 is used to access a memory location in the next pointer RAM 106. The NextValid value is stored in that accessed location. Fig. 3 reflects this action in the graphic identified by 208.

[29]   In a step 210, the NextValid value is then loaded into the selected TOQ register. Completing the discussion of Fig. 2, step 212 is performed to increment the LL register corresponding to the QID to indicate that the specified queue now contains one more entry.

[30]   Refer now to Figs. 5, 6, and 7 for a discussion of a read operation in accordance with the embodiment of the invention shown in Fig. 1. Fig. 5 is a high level flow chart, highlighting the steps comprising a read operation on a queue. Fig. 6 graphically represents the data flow among the elements of the circuit shown in Fig. 1 during a read operation. Fig. 7 highlights the data flow in the circuit diagram of Fig. 1 during a read operation. It can be appreciated that the decoder logic 134 and the control logic 132 can be further configured to perform the operations that will be discussed in connection with a read operation.

[31]   When a request to read a queue is communicated to the circuit of Fig. 1, the request comprises a QID and an indication of the read operation. In a step 502, the register file 102 is accessed to access the HOQ register 114 corresponding to the specified queue (see also Fig. 6). In a step 504, the memory location in the buffer RAM 104 pointed to by the content of the HOQ register is accessed and read out on a data bus 174 of the buffer RAM.

[32]   In a step 506, the content of the HOQ register is used to access a memory location in the next pointer RAM 106. The content of the accessed memory location in the next pointer RAM represents the next entry in the specified queue. Fig. 6 shows this value as being identified by NextValid. Then in a step 508, the content of the HOQ register is pushed onto

7

the queue in the free pointer RAM 108, thus freeing the location in the buffer RAM 104 that was just read.

[33]   The NextValid value is then loaded into the HOQ register, step 510. To complete the flow chart of Fig. 5, the LL register corresponding to the specified queue is decremented by one to indicate there is now one less entry in that queue.

[34]   It is worth noting that the control logic 132 can be configured to track how much space remains in the buffer RAM 104 to detect when the memory is full. This can be used to trigger an overflow condition to indicate there is no more space for a new queue entry. It can be appreciated that any of a number of techniques can be used to detect a "memory full" condition. For example, the top of queue and bottom of queue can be monitored. A global counter can be used and checked each time a write operation is detected, and incremented after a successful write operation. The individual LL registers can be summed each time a write operation is detected.

[35]   The control logic can be further configured to detect an attempt to read an empty queue. This may not be necessary if it can be assumed that the system in which the invention is incorporated will behave correctly and not make an attempt to read an empty queue.

[36]   Figs. 8A - 8J show a sequence of write operations and read operations to illustrate the operation of this particular embodiment of the invention. Fig. 8A represents the initialized state of the circuit of Fig. 1. Notably, the free pointer RAM (fRAM) is initialized as discussed above. The top of queue (T) points to the first available memory location in the buffer RAM (bRAM), and the bottom of queue (B) points to the last available memory location. The next pointer RAM (nRAM) and the bRAM may or may not be initialized. suppose the following operations are performed:

write "data1" to queue 1 (Q1)
write "data2" to Q1
write "data3" to queue 3 (Q3)
write "data4" to Q1
write "data5" to Q3
read from Q1
read from Q3
read from Q3
write "data6" to Q3

[37]   The first write operation is a write operation of data (data1) to Q1. Fig. 8B summarizes the data flow for this write operation. The TOQ register for Q1 is accessed. The content of the register T1 contains the value "1" per the initialization sequence discussed

8

above. In accordance with step 204 (Fig. 2), memory location "1" in the bRAM is written; the notation "bRAM(1) ← 1" will be used to represent the action. The fRAM is then popped (step 206); this will produce the value 16, since it was at the top of the queue. As can be seen in Fig. 8B, after the pop operation, the top of the queue moves down to the next entry in the fRAM queue, and now points to "17" in the fRAM. The value ("16") that was popped represents the memory location in the buffer RAM for data of the next entry in Q1. The popped value is stored in the next pointer RAM at the memory location determined based on the content of the T1 register (TOQ register for Q1), step 208. The popped value is then loaded into the T1 register, 210. It is understood that the corresponding link length counter for Q1 is incremented.

[38]   The next operation is a write operation of "data2" into Q1. Fig. 8C shows the data flow. Again, the T1 register is accessed, and this time the content of the T1 register is "16." The data is therefore stored in memory location 16 of bRAM. Again, the fRAM is popped to obtain the memory location in the bRAM for the next entry for Q1; this time the value is "17." The memory location 16 in the nRAM is loaded with the value "17," and likewise the T1 register is loaded with "17."

[39]   The next write operation illustrated in Fig. 8D is a write of "data3" to Q3. The initial value contained in the TOQ register for Q3 is "3" per the initialization sequence discussed above. Hence, "data3" will be written to memory location 3 in the bRAM. The next value popped from the fRAM is "18," which is then stored in memory location 3 in the nRAM. The T3 register is then loaded with the value "18."

[40]   Fig. 8E shows the details for the write operation of "data4" to Q1.

[41]   Fig. 8F shows the detail for the write operation of "data5" to Q3.

[42]   A read operation is then performed on Q1. As can be seen in Fig. 8G, the HOQ register for Q1 is accessed, in accordance with step 502 of Fig. 5. Since this is the first read operation on Q1, it contains the initial value of "1". This references the memory location 1 in bRAM, which from Fig. 8B, stores the data for the first entry in Q1. The data is therefore read from bRAM(1), step 504. Next, a memory location in the nRAM identified by the H1 register (HOQ register for Q1) is accessed, step 506. Recall from Fig. 8B that this memory location contains the memory location in the bRAM which contains the data for the next entry in Q1. Next, the value contained in the H1 register is pushed onto the fRAM, thus freeing the memory location in the bRAM identified by the value in the H1 register. Fig. 8G shows the fRAM now contains the value "1" and the bottom of queue (B) points to "1."

[43] Fig. 8H shows that another read operation is performed, this time on Q3. The HOQ register (H3) for Q3 is accessed. The H3 register contains its initial value of "3" and so the data in bRAM(3) is read out. The memory location in nRAM(3) is accessed to produce the value "18." As can be seen from the write operation shown in Fig. 8D, the value "18" is the memory location in bRAM that contains that data for the next entry in Q3. The value ("3") stored in the H3 register is pushed onto the queue in the fRAM. The H3 register is then loaded with "18," the value obtained from the nRAM.

[44] Fig. 8I shows the details for another read operation on Q3. This read operation empties the queue. It is worth noting that the H3 register is loaded with the value "20." However, since the queue is empty, a subsequent read attempt should not be permitted. This can be enforced by the control logic 132, or at a higher level component in a system that incorporates the invention. Referring to Fig. 8F, it can be seen that a write operation to Q3 will place the data in memory location "20" of the bRAM.

[45] Fig. 8J shows a write operation of "data6" to Q1.

[46] It can be appreciated from the foregoing sequence of write operations and read operations, that the queue sizes are not statically determined. Each queue can grow as much as needed, so long as there is sufficient buffer RAM 104. Memory allocated to a queue is immediately released to a free memory pool maintained by the free pointer RAM, so that the memory can be allocated to other queues. Speed of operation for allocating and deallocating memory for the queues is greatly enhanced by having a memory for storing the queue pointers, and a memory to track the free memory pool and a separate memory for maintaining the linkage information for the queues. The invention allows for scaling up to larger numbers of queues (i.e., N > 16) without affecting performance in queue access time.